Metamorphic Testing of Navigation Software: A Pilot Study with Google Maps

Joshua Brown, Zhi Quan Zhou†, and Yang-Wai Chow
Institute of Cybersecurity and Cryptology
School of Computing and Information Technology
University of Wollongong
Wollongong, NSW 2522, Australia

***Abstract*—Millions of people use navigation software every day to commute and travel. In addition, many systems rely upon the correctness of navigation software to function, ranging from directions applications to self-driving machinery. Navigation software is difficult to test because it is hard or very expensive to evaluate its output. This difficulty is generally known as the oracle problem, a fundamental challenge in software testing. In this study, we propose a metamorphic testing strategy to alleviate the oracle problem in testing navigation software, and conduct a case study by testing the Google Maps mobile app, its web service API, and its graphical user interface. The results show that our strategy is effective with the detection of several real-life bugs in Google Maps. This study is the first work on automated testing of navigation software with the detection of real-life bugs.**

Keywords: Mobile navigation software, Google Maps, directions API, Web service, Graphical User Interface, software testing, oracle problem, metamorphic testing, verification, validation.

1. Introduction

The road network is a vast and complex system of interconnecting roads and intersections which are extended through the use of external services such as a road-ferry. In the world, there are over 64 million kilometres of road [1] and it continues to grow and change every day. Navigation software is designed to plan an optimal route between two points within the constraints it has been passed; the route contains the directions that the user should follow to reach the destination. Navigation systems are one of the most popular applications on the Internet, and on smart phones. They are the most popular application installed on over 50% of the global smart phone market [2]. Furthermore, these systems are mission critical applications, as their failure could potentially cause traffic accidents, especially when they are used for the navigation of self-driving cars. Navigation systems, therefore, must be thoroughly verified and validated.

To verify and validate software systems, testing is essential. It is widely accepted that, in a typical commercial software development project, the cost of testing can easily exceed 50% of the total development budget. Testing involves executing the *software under test* (SUT) with a set of test cases together with a mechanism against which the tester can decide whether the outcomes of test case executions are correct (that is, a *test oracle*). The *oracle problem* refers to the situation where an oracle does not exist or it is theoretically available but practically too expensive to be applied. The oracle problem is a fundamental challenge in software testing practice but this problem is often ignored by the research community — compared with many other aspects of testing such as automated test case generation, the challenge of test oracle automation "has received significantly less attention, and remains comparatively less well-solved" [3].

There is a severe oracle problem when testing navigation software. This is because the real-world road networks are so complex that in most situations it is infeasible to validate whether or not a route returned by the navigation software is correct and optimal, except trivial cases. This difficulty results in a scenario where developers cannot utilize conventional testing techniques. In fact, there has been little research on automated testing of navigation software in the literature. A related study was conducted by Wright et al. [4], where they manually evaluated the position accuracy and measurement accuracy (driving distance) computed by GPS receivers against a local area road map.

A growing body of research and industrial application has investigated the concept of *metamorphic testing* (MT) [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], and has proven MT to be a highly effective testing paradigm for the detection of real-life software faults in the absence of an ideal test oracle. The idea of MT is simple: Instead of focusing on the correctness of each individual output, MT looks at the relationships among the inputs and outputs of *multiple* executions of the SUT. Such relationships are called *metamorphic relations* (MRs), and are necessary properties of the intended software's functionality. For example, when testing a web search engine, it is very hard for testers to assess the quality of the search results, such as their accuracy and completeness [8]. Nevertheless, MT can be applied by identifying the following metamorphic relation: A stable search engine should return similar results for similar queries. For instance, although a search for [`today's movies in Honolulu`] and a search for [`Honolulu movies today`] may return different results, the two

†*Corresponding author. Email: zhiquan@uow.edu.au. Telephone: (+61-2) 4221 5399.*

URI: http://hdl.handle.net/10125/50602
ISBN: 978-0-9981331-1-9

sets of search results should have a large intersection if the search engine under test is robust [8]. Because MT looks at the relationships among multiple SUT executions instead of focusing on the verification of each individual output, it can be performed in the absence of an ideal oracle, hence alleviating the oracle problem.

Although the basic concept of MT is simple, it requires specific study when applied to different application domains [9]. This is because different application areas can have different properties of interest to investigate. The present paper proposes applying MT to test navigation software in order to alleviate the oracle problem in testing such systems. A case study has been conducted with Google Maps, including the testing of its mobile applications, its web service APIs (namely, the Directions API), as well as its Graphical User Interface (GUI) at maps.google.com. Google Maps was selected for the case study because it is the most popular mapping system by far [2] (except in China, where Google services could not be accessed).

In this study we ask the following research question:

- RQ: Can we have a practical and effective method of automatically testing navigation systems in the face of the severe oracle problem?

The contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work in the literature to address *automated* testing of navigation software.
- The detection of real-life bugs in Google Maps is significant, and demonstrates the effectiveness of our approach.

The testing methodology presented in this study can also be applied to other forms of navigation systems beyond Google Maps, and can be used as the foundation for future work in the verification and validation of similar systems such as self-driving cars.

The rest of this paper is organized as follows: Section 2 further introduces some background knowledge and describes our testing approach with a focus on the identified MRs for the navigation software under test. Section 3 presents our test results by highlighting the detected defects in Google Maps. Section 4 makes further discussions and concludes the paper.

2. Our Approach for Testing Navigation Software

In this section, we first look at the difficulties in testing navigation software, and then describe our testing approach.

2.1. Difficulties in Testing Navigation Systems

As explained in Section 1, navigation systems are difficult to test owing to the oracle problem. Worse, the lack of system specifications adds further difficulties to user validation: The vast majority of users do not have access to the detailed algorithm designs and system/subsystem specifications of the navigation software they are using. Without access to these specifications, the user manual or online help pages are the only type of information source available to the users. However, as pointed out by Zhou et al. [8], user manuals or online help pages are usually very brief and are not equivalent to the system specification defined as "an adequate guide to building a product that will fulfill its goals" [15]. Consequently, it is basically impossible for users to evaluate the navigation system they are using against its technical specifications or the intended algorithms.

Zhou et al. [8] pointed out that the above phenomenon can be quite common when testing many types of software applications such as web services, poorly evolved software, and open source software, and showed that MT can be an effective approach to addressing these difficulties caused by a lack of detailed knowledge about the system design and specifications coupled with the oracle problem.

For developers of the navigation systems, even if they have complete documentation and specifications, it is still very challenging for them to test such systems because of the complexity of the underlying algorithms and data. As will be shown in this paper, MT can be an effective testing approach for both developers and users, as well as third-party independent testers.

2.2. Metamorphic Testing (MT)

MT [5], [8], [9] alleviates the oracle problem by testing the SUT against prescribed MRs, which are necessary properties of the intended program's behavior. The difference between MRs and other types of program correctness properties is that an MR involves *multiple* executions of the target program. Even if the correctness of an individual output cannot be verified due to the lack of an oracle, the tester can still check whether the expected relationship among multiple executions is satisfied. If the MR is *violated* for any of the test cases, a *failure* is detected.

For example, let $p(G,\ x,\ y)$ be a program that calculates the shortest path from node x to node y in an undirected graph G. When G is large and complex, it can be difficult to verify the output of p because no oracle can be practically applied. To perform MT in this situation, we can identify many MRs for the shortest path problem, one of which can state that swapping the origin and destination nodes should not affect the length of the calculated shortest path [16]. Using this MR, a metamorphic test will run p twice, namely, a *source execution* on a *source input* $(G_1,\ x_1,\ y_1)$ to produce a *source output*, and a *follow-up execution* on a *follow-up input* $(G_2,\ x_2,\ y_2)$ to produce a *follow-up output*, where $G_2 = G_1$, $x_2 = y_1$, and $y_2 = x_1$. Any violation of this MR (that is, if the source output and the follow-up output are found to have different lengths for some test case(s)) will reveal a fault in p. Many other MRs can also be identified and used to test p [16], and different MRs often have different fault-detection capabilities.

When MT was first proposed, it was designed as a *verification* technique, where an MR is a necessary property of the intended algorithm or system specification to be implemented. In this situation, a violation of the MR reveals a fault in the implementation. Recently, Zhou et al. [8] pointed out that MRs can also be defined based on user expectations "to reflect what they really care about," rather than based on the algorithms or system specifications of the developers—such algorithms and specifications are often unknown to the users anyway. In this way, MT can be used as a user-oriented approach to perform *validation* and other types of *quality assessment* (such as the assessment of *usability* and *functional completeness*), and hence MT has been developed into a unified framework for software verification, validation, and quality assessment [8].

2.3. The Identified Metamorphic Relations for Navigation Software

In this research, we adopt a user-orientated testing approach by utilizing the concept of MT, as we identify MRs for navigation software from a user's perspective. This is because the algorithms and detailed system specifications of the SUT are unavailable for reference. Although this pilot study has a limited test scope of Google Maps, the approach proposed here can be applied to test other navigation systems.

A total of four MRs have been identified, as described in the subsections below.

2.3.1. MRSimilar. The first MR is named MRSimilar. Its design is based on the premise that a navigation system should return similar results for similar queries, in a way similar to a search engine [8].

For instance, after a source output (a route) is generated for a source input (an origin and a destination point), we can produce a follow-up input by very slightly changing the origin and/or the destination. Then, in most situations, the follow-up output should be a route having a cost similar to that of the source output. In this paper, the *cost* of a route is in terms of distance or time depending on the user's preference; monetary costs are not considered.

More specifically, let $d(a,\ b)$ be a function that gives the cost of an optimal route for travel from point a to point b. MRSimilar states that $d(a,\ b)$ and $d(a',\ b')$ should be similar if $a \approx a'$ and $b \approx b'$. Here, $x \approx y$ means that x and y are approximately at the same location.

In our experiments with MRSimilar, each source test case (namely, an origin and a destination point) was formed by means of random selection from a set of addresses (to be explained in Section 2.4), and the corresponding follow-up test case was produced by adding a tiny amount of distance (e.g., a few millimeters or centimeters on the same road) to the origin and/or destination point. A comparison was then made assessing the difference between the source and the follow-up outputs. An anomaly would be reported if a large amount of difference (e.g., more than ten meters) was detected.

2.3.2. MRRestriction. The second MR is named MRRestriction. It employs the navigation system's ability to work under different conditions. Examples include avoiding elements of the route such as tolls, ferries, and highways. The MR is focused on ensuring that a restrictive condition does not result in a more desirable/optimal output. More specifically, MRRestriction states that

$$d_R(a,\ b) \geq d(a,\ b),$$

where $d_R(a,\ b)$ is a function that gives the cost (distance or time) of an optimal route for travel from a to b with a restriction, such as avoiding highways.

MRRestriction can be used to assess how the output of a navigation system is affected by the rules placed on the user request or affected by external conditions under which certain elements of the route are not available (such as outside ferry operating hours). It is based on the concept that a query without any restriction should yield a more beneficial result than a query that has restrictive rules on it. For example, a route without any restriction should not be longer and slower than a route that avoids highways.

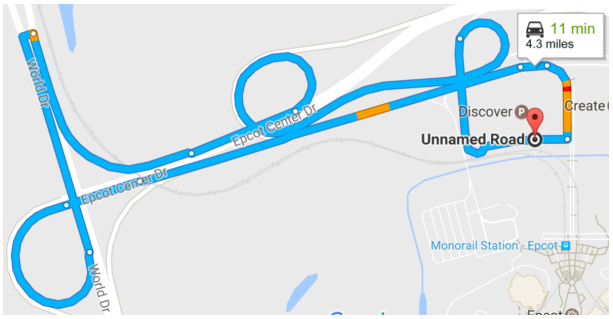
In the experiments with MRRestriction, each metamorphic test started with a source test case and then a follow-up test case was constructed with the addition of restrictions such as avoiding ferries, tolls, highways, or a combination of these restrictions. If a follow-up output was more optimal than the source output, an anomaly would be reported. A restriction can be added *explicitly* by selecting certain options in the user query or *implicitly* by setting a travel time outside certain road/ferry/bus/train operating hours.

2.3.3. MRSplit. The third MR is named MRSplit. It observes that the cost of a route from a to c via b should be similar to the cost of a route from a to b plus the cost of a route from b to c. More generally, MRSplit requires that $d_m(a_1,\ a_2,\ \ldots,\ a_n)$ should be similar to $d(a_1,\ a_2) + d(a_2,\ a_3) + \ldots + d(a_{n-1},\ a_n)$, where $d_m(a_1,\ a_2,\ \ldots,\ a_n)$ denotes the cost of an optimal route for travel from a_1 to a_n via a_2, a_3, …, a_{n-1}.

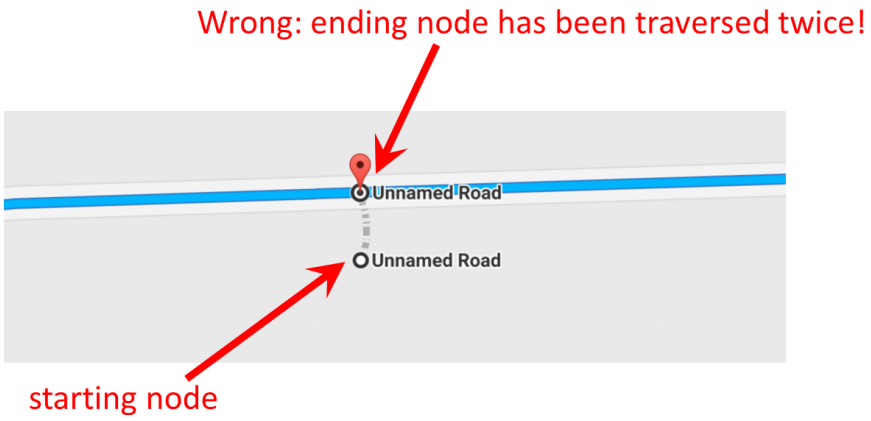
MRSplit can be used to assess how the output of the navigation software is affected by intermediate nodes. In our experiments, each source test case included some waypoint nodes between the origin and destination, and a series of follow-up test cases were formed by splitting up the source test case.

2.3.4. MREnvironment. Our last MR is named MREnvironment. It assesses how the navigation system's behavior is affected by different user environments. An example of this MR is the same request issued using the API (source input) and the mobile application (follow-up input)—an ideal navigation system should return similar results across these different user environments.

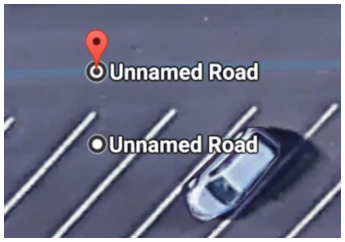
More specifically, let P and Q be two different environments or platforms. Let d_P and d_Q be the cost functions for environments P and Q, respectively. MREnvironment states that $d_P(a,\ b)$ and $d_Q(a,\ b)$ should be similar.



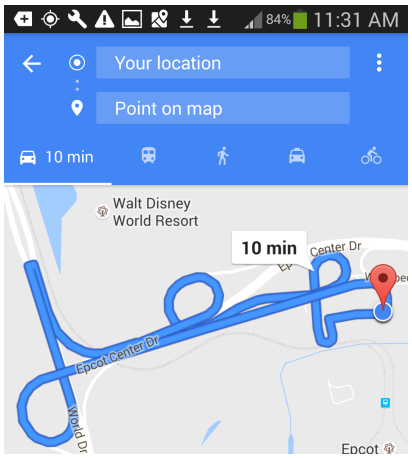
(a)



(b)



(c)



(d)

Figure 1. Google Maps failure detected using MRSimilar with American addresses. (a) The entire driving direction generated by Google Maps (screenshot taken at maps.google.com). (b) Zoom in to show that the destination point has been traversed twice in the route. (c) Further zoom in to show the origin and destination points. (d) A similar failure produced by the Google Maps app for Android on a Samsung Galaxy S4 mobile phone.

In our experiments with MREnvironment, each metamorphic test consisted of source and a follow-up test cases involving exactly the same query but different environments. An anomaly would be reported if the outputs were significantly different (e.g., having more than ten meters difference).

2.4. General Design of the Experiments

For the experiments, a set of source test data was generated by randomly sampling a large number of addresses in Australia and a small number of addresses in America, as it was more convenient for the authors to validate Australian addresses.

An individual address formed the basis of a node. Each request to the navigation software can be broken down into five elements: a starting node, an ending node, waypoint node(s) (that is, the intermediate stop(s)), time (departure or arrival time), and restrictions.

To ensure a stable and consistent testing environment, the real-time traffic feature of Google Maps was turned off during the experiments, as otherwise routing might be affected by live traffic and the test results might not be repeatable. To avoid personalized results, the tester did not log into any online accounts including Google accounts. Furthermore, when an anomaly was observed, the test was immediately repeated. An anomaly would be reported only if it could be reproduced. This treatment was to ensure that the reported anomalies were not caused by the update or dynamics of the maps or algorithms.

Apart from the identification of the MRs (which cost about one hour brainstorming), the testing process was largely automated by means of test scripts and test drivers. In this research, a total of 1,000 hours of test executions were completed across the different environments of Google Maps. As all of the tests were web-based, the impact of hardware selection was very small.

3. Issues Detected in Google Maps

Overall, Google Maps passed the majority of the executed tests; however, there were cases where the system resulted in an unexpected output as detected by MR violations. A manual inspection of the MR violations revealed several defects in Google Maps. This section will highlight some notable examples of these defects.

3.1. Defects Detected by MRSimilar

In this subsection, we report two failures detected by MRSimilar, where the first failure was detected when searching for a route in America and the second failure was detected when searching for a route in Australia. In software testing, a *failure* refers to erroneous behavior of the SUT.

3.1.1. Searching for a route in America. When Google Maps was tested against MRSimilar, a source test case yielded an output that had an expected time duration of 1 minute and a distance of 0.0 miles. The follow-up test case featured the same starting node and the same conditions and only modified the ending node by a distance of 0.98 cm; the follow-up output, however, suddenly changed dramatically with an expected time duration of 11 minutes and a distance of 4.3 miles, as shown in Figure 1.

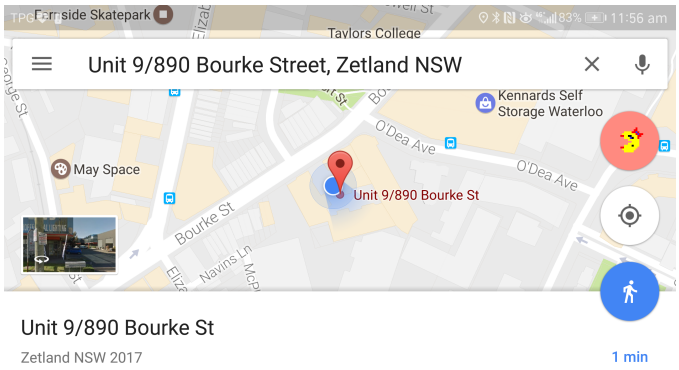
Figure 1 (a) shows the entire driving direction generated by Google Maps. The screenshot was taken by using a browser to access the website maps.google.com. Figure 1 (b) zooms in to show a portion of the route surrounding the origin and destination points. It is surprising to see that the destination point has been traversed twice in the driving direction, which is an obvious error. Figure 1 (c) zooms in further to show a Satellite View of the exact origin and destination points. Figure 1 (d) is an excerpt of a screenshot taken from a Samsung Galaxy S4 mobile phone running the Google Maps app for Android. This screenshot shows that a similar failure was produced using the mobile device.

Given the extremely small distance between the origin and destination points shown in Figure 1, a normal driver would probably not require any navigation software to guide him or her. It is, however, not the case for self-driving vehicles or robots because such autonomous machinery will always rely on software systems to navigate. It is not acceptable for a driverless car to travel 4.3 miles to reach a destination that is actually only two meters ahead.

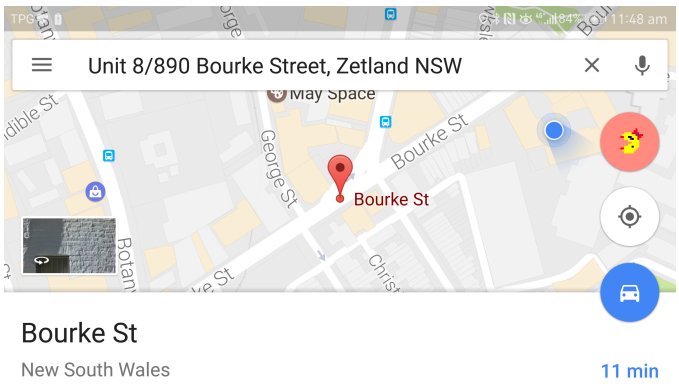
3.1.2. Searching for a route in Australia. Figure 2 shows another failure detected by MRSimilar, using Australian addresses. Figure 2 (a) shows a screenshot taken from a Huawei mobile phone running the Google Maps app for Android. It shows that Google Maps returned 1 minute walking distance from the current location to "Unit 9/890 Bourke Street, Zetland NSW."

To conduct MT, we also queried the Google Maps app using a slightly modified destination address, which differed from the previous address only in the unit number (namely, using "Unit 8" instead of "Unit 9," as we verified that "Unit 8" was a valid address), but Google Maps returned a very different location that was 2 km away, which required 11 minutes driving (Figure 2 (b)) or 26 minutes walking (Figure 2 (c)). A violation of MRSimilar was reported because the difference between the "Unit 8" route and the "Unit 9" route was too large given that they were at the same street address.

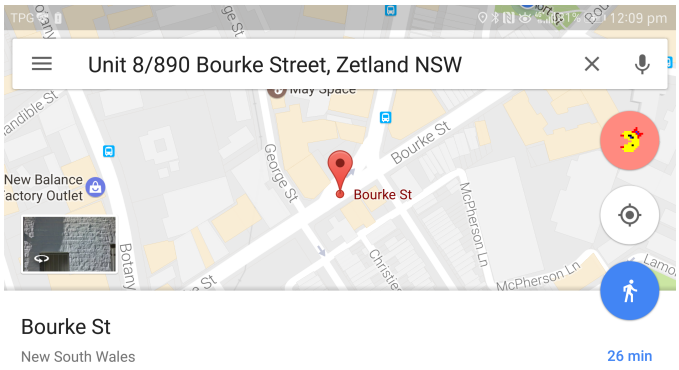
After the MR violation was reported, we manually analyzed the test results to investigate the root cause of the failure. We first validated that "Unit 8/890 Bourke Street, Zetland NSW" was indeed a correct address and that Unit 8 and the other units were physically located near each other at 890 Bourke Street. Figure 3 shows a picture taken at the entrance to Unit 8 during the site visit. We further found that Google could not actually locate this unit (although it had been a valid address for a long time) and therefore *automatically* changed the user query from "Unit 8/890 Bourke Street, Zetland NSW" to "Bourke Street, Redfern, NSW" without explicitly requesting the user to confirm the



(a)



(b)



(c)

Figure 2. A Google Maps app failure on a Huawei Mate 9 Pro mobile phone running Android, detected using MRSimilar with Australian addresses. (a) Google Maps app returned "1 min" for walking from the current location to "Unit 9/890 Bourke Street, Zetland NSW." (b) Google Maps app returned "11 min" for driving from the current location to "Unit 8/890 Bourke Street, Zetland NSW." (c) Google Maps app returned "26 min" for walking from the current location to "Unit 8/890 Bourke Street, Zetland NSW." In (a), (b), and (c), the "current location" was basically the same. It was found that the location for the address "Unit 8/890 Bourke Street, Zetland NSW" generated by the Google Maps app in (b) and (c) was wrong: It was 2 km away from the actual location.

Figure 3. Site visit to Unit 8/890 Bourke Street, Zetland, NSW, Australia, which confirmed that this was a valid physical address and that Unit 8 and the other units were located near each other.

change. In this modified address, the unit number "8" and the street number "890" were removed, and the suburb "Zetland" was changed to "Redfern." This explains why the location returned by the Google Maps app was 2 km away.

We recognize that many so called "intelligent" systems (actually, their developers) might assume that they were smarter than the human users and therefore could automatically "correct" user input without even informing the users. This phenomenon was initially reported by Zhou et al. [8] where they studied web search engines and found that major search engines could automatically change user queries without explicitly informing the users, hence revealing a crucial deficiency in the software system's functional completeness.

3.2. Defects Detected by MRRestriction

Figure 4 shows a failure where Google Maps generated an infeasible route involving vehicular ferries. In Figure 4 (a), the user searched for a route and set the departing time to be "5:00 AM." Google Maps returned a route that will "arrive around 5:30 AM." The route involved the use of free vehicular ferry service operated by the government, which carries cars across the Clarence River. According to the government official website (Figure 4 (b)), the ferry operating hours start at 6:00 AM seven days a week. This means that the route shown in Figure 4 (a) is infeasible for the user's travel time (departing at 5:00 AM and arriving at the destination around 5:30 AM). Google Maps failed to identify this restriction and still recommended the user to use the (unavailable) ferry service. Actually, there was a nearby bridge that should be recommended instead for the given travel time.

This failure could be potentially due to incorrect ferry operating time data in the Google Maps database.

3.3. Defects Detected by MRSplit

When testing the Google Maps API against MRSplit, one of the source test cases involved an origin, a destination, and eight intermediate nodes. The corresponding follow-up test cases consisted of the nodes broken up in individual queries. For the source test case, Google Maps API returned UNKNOWN_ERROR as follows:

```
{
"routes" : [],
"status" : "UNKNOWN_ERROR"
}
```

In Google Maps API online documentation, an UNKNOWN_ERROR indicates that "a directions request could not be processed due to a server error" [17]. The "smaller" follow-up test cases, however, did not result in any error.

To further investigate this issue, we ran the test via the website GUI at maps.google.com, and the test passed without causing any failure or error, as shown in Figure 5. This observation demonstrates that the Google Maps GUI and API are actually not the same, and in this test the API appeared to be more vulnerable to "large" input involving a large number of waypoint nodes. A further investigation shows that this failure could also be replicated using other "large" inputs.

3.4. Defects Detected by MREnvironment

More failures were detected that were unique to the API environment, an example of which is shown in Figure 6, where a problem related to Geocoding was revealed. Geocoding is a process of converting addresses into geographic coordinates. Figure 6 shows that, for the test case under consideration, the API could not find the address, and therefore could not generate the geographic coordinates. As a result, the API could not generate a route, hence reporting a "NOT_FOUND" status.

When the same query was made via the Google Maps website GUI, it generated a route without causing any problem, as shown in Figure 7. This obserevation again suggests that the Google Maps API was not as reliable as the website GUI.

4. Discussions and Conclusion

In Section 1, we asked a research question: Can we have a practical and effective method of automatically testing navigation systems despite the oracle problem?

To meet this challenge, we proposed applying MT to test navigation systems, and have completed a pilot study

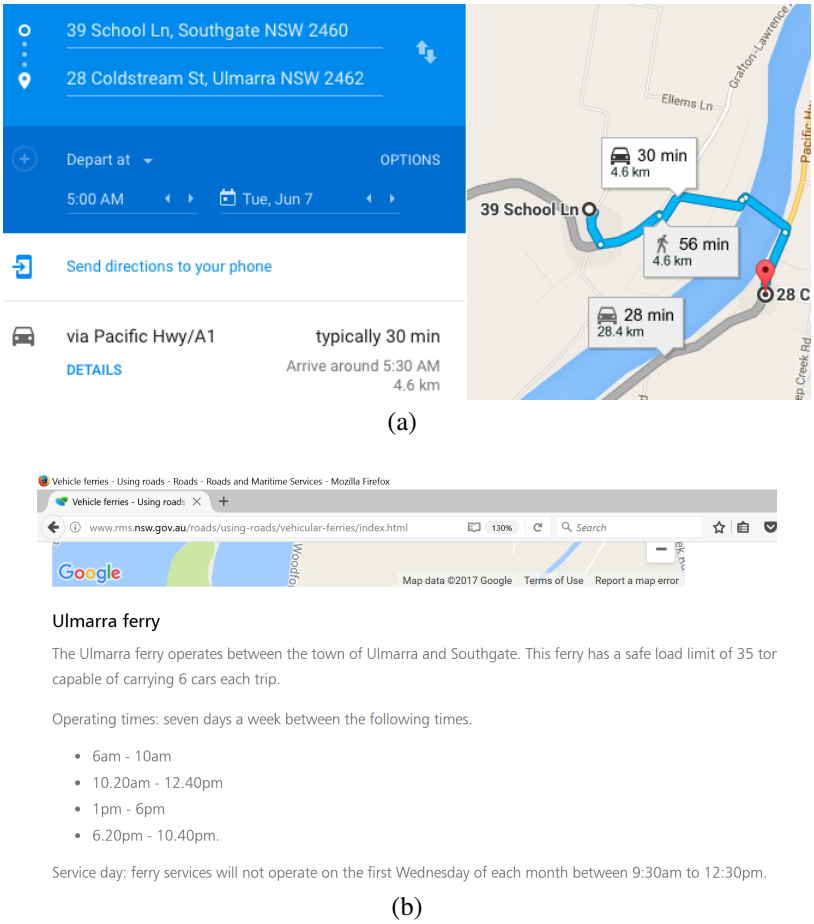


Figure 4. Google Maps returned an infeasible route. (a) Excerpts of a screenshot showing that Google Maps returned a route involving vehicular ferry service for travel departing at "5:00 AM" and arriving "around 5:30 AM." (b) Ferry operating times: It starts at 6:00 AM, seven days a week.

using Google Maps. The results of this study provide an affirmative response to the research question. The detection of several types of real-life bugs in Google Maps further demonstrated the effectiveness of MT in testing "non-testable programs," i.e. programs that are difficult to test due to the lack of an oracle. Our testing approach can be used by developers for software verification, by users for software validation, and by independent testers for various quality assessment purposes.

Compared with the significance of the detection of major defects across several different environments (namely, the Google Maps mobile app, its web service API, and its GUI at maps.google.com), the testing cost was relatively small. The reported failures could be caused by problems in the routing algorithms and/or the underlying databases. We have reported our findings to Google, and received a reply indicating that these issues are currently being investigated.

This pilot study suggests that navigation systems can be considered as a special type of search engine, which accepts user queries and returns routes or driving directions. Previous results on search engine testing [8] can therefore be useful for the testing of navigation systems. This research employed a useful general metamorphic relation (that is, a pattern of metamorphic relation) that is valid for both search engines and navigation systems, namely, the software under test should return similar results for similar queries. This kind of general MR, or MR pattern, can be used to derive many concrete MRs. In future research, more effort should be made into the identification of MR patterns that can be used across different application domains.

This research can also be significant for the development of testing techniques for the navigation components of self-driving vehicles and even self-navigating drones. Future research will be conducted at a larger scale by taking these systems into consideration.

Acknowledgments

This work was supported in part by a linkage grant of the Australian Research Council (project ID: LP160101691) and an Australian Government Research Training Program scholarship.

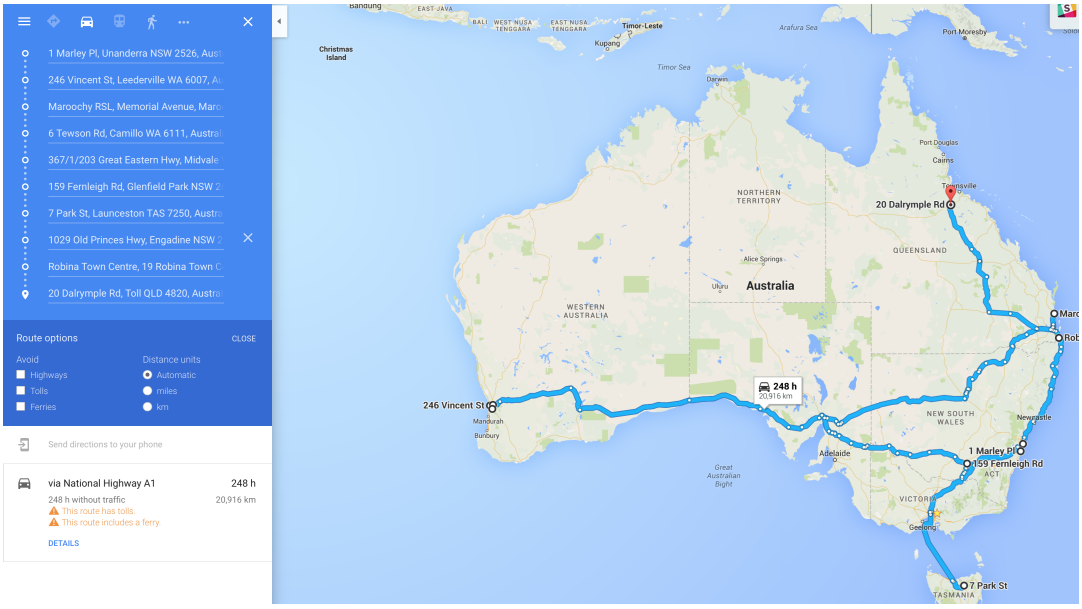


Figure 5. The Google Maps website GUI at maps.google.com successfully passed this test case that had eight intermediate nodes, while the API failed.


```
"geocoded_waypoints" : [
{
"geocoder_status" : "ZERO_RESULTS"
},
{
"geocoder_status" : "OK",
"place_id" : "ChIJd0ShQqAZE2sRLCpaDBjrqUY",
"types" : [ "street_address" ]
}
],
"routes" : [],
"status" : "NOT_FOUND"
}
```


Figure 6. Google Maps API failure: Geocode not found.

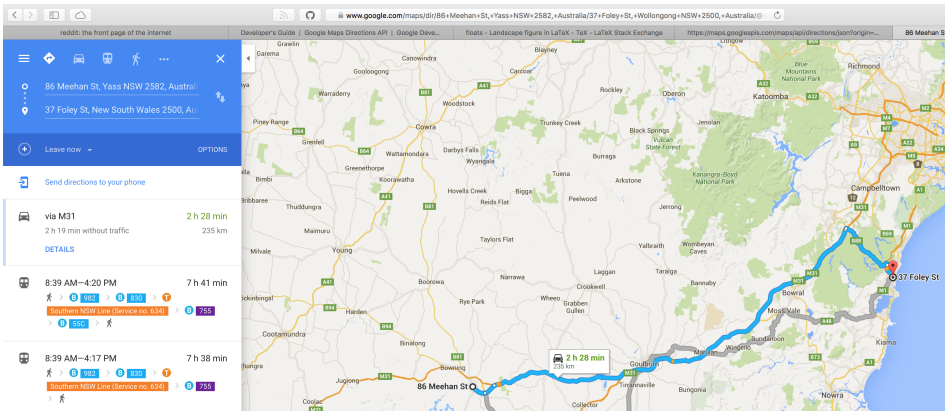


Figure 7. For the same test case, the Google Maps website GUI at maps.google.com has passed (as shown in this figure) whereas the Google Maps API failed (as shown in Figure 6). This problem was detected using MREnvironment.

References

[1] "The world factbook," Central Intelligence Agency, 2013. [Online]. Available: https://www.cia.gov/library/publications/the-world-factbook/fields/2085.html

[2] BuiltWith, "Mapping usage statistics," 2017. [Online]. Available: https://trends.builtwith.com/mapping

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[4] M. Wright, D. Stallings, and D. Dunn, "The effectiveness of global positioning system electronic navigation," in *Proceedings of IEEE SoutheastCon*, 2003, pp. 62–67.

[5] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 1, pp. 1–9, 2003.

[6] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, 2014.

[7] M. Lindvall, D. Ganesan, R. Árdal, and R. E. Wiegand, "Metamorphic model-based testing applied on NASA DAT – an experience report," in *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, 2015, pp. 129–138.

[8] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, 2016.

[9] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[10] T. Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, and Z. Q. Zhou, "Metamorphic testing for cybersecurity," *Computer*, vol. 49, no. 6, pp. 48–55, 2016.

[11] U. Kanewala, L. L. Pullum, S. Segura, D. Towey, and Z. Q. Zhou, "Message from the workshop chairs," in *Proceedings of the IEEE/ACM 1st International Workshop on Metamorphic Testing (ICSE MET'16), in conjunction with the 38th International Conference on Software Engineering (ICSE)*. ACM Press, 2016.

[12] D. C. Jarman, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing for Adobe data analytics software," in *Proceedings of the IEEE/ACM 2nd International Workshop on Metamorphic Testing (ICSE MET'17), in conjunction with the 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 21–27.

[13] J. Ding, X.-H. Hu, and V. Gudivada, "A machine learning based framework for verification and validation of massive scale image data," *IEEE Transactions on Big Data*, in press.

[14] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, in press.

[15] M. Pezzè and M. Young, *Software Testing and Analysis: Process, Principles, and Techniques*. New York: Wiley, 2008.

[16] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC'04)*. Polytechnic University of Madrid, 2004, pp. 569–583.

[17] "Google Maps Directions API," Google, 2016. [Online]. Available: https://developers.google.com/maps/documentation/directions/intro